USN | | | | | | | | | |

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – September 2020

Sub:	CAD of Digital Systems				Sub Code:	18EVE31	Branch:	M.Tech
Date:	15/09/2020	Duration:	90 min’s	Max Marks:	50	Sem / Sec:	III, VLSI	OBE

Answer any FIVE FULL Questions

	MARKS	CO	RBT
1. What are the important design (Optimization) entities in VLSI Design? Explain.	[10]	CO1	L1
2. Enumerate and explain the VLSI design methods and technologies.	[10]	CO1	L1
3. Discuss the domains and their hierarchies in VLSI design with Y Chart.	[10]	CO1	L2
4. Explain briefly about the data structures for the representation of Graphs.	[10]	CO1	L1
5. Explain with a pseudo code and example about the depth first search algorithm.	[10]	CO1	L3
6. Draw the Y-Chart showing the VLSI design tools and explain them in detail.	[10]	CO1	L2
7. Explain with an example the breadth-first search algorithm with a Pseudo-code	[10]	CO1	L3

CMR Institute of Technology
Department of ECE
M.Tech III Sem VLSI Design & Embedded Systems
IAT -1 Scheme and Solution
18EVE31- CAD of Digital Systems

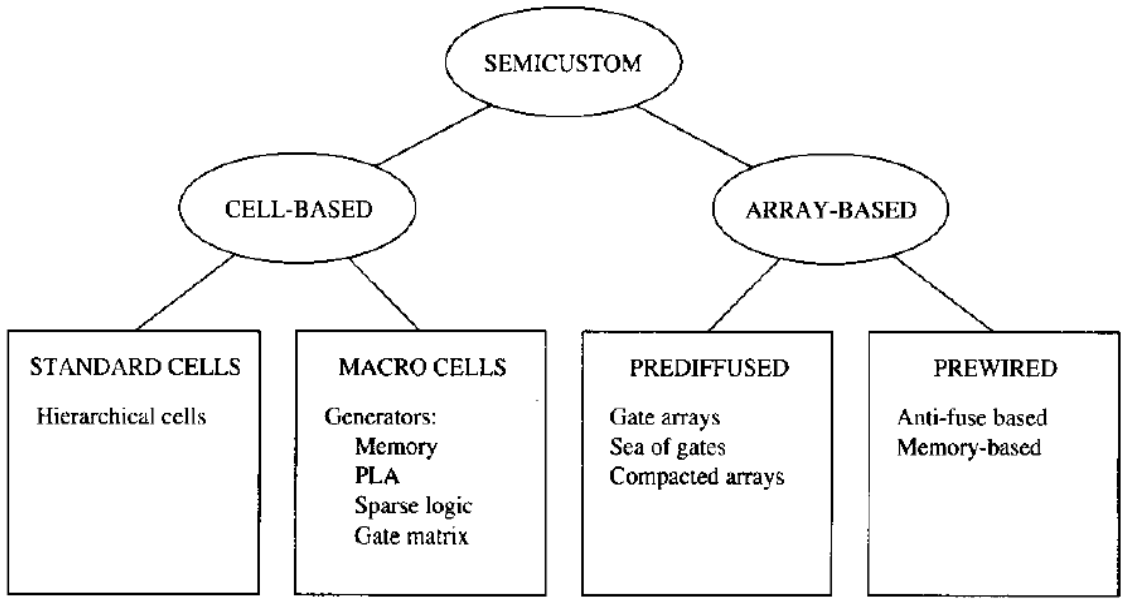
1. What are the important design (Optimization) entities in VLSI Design? Explain.

Several entities- cannot be optimized simultaneously; one can be improved at the expense of one or more

- ➢ **Area**: minimization is required since less silicon is used ,but also the yield is increased
- Yield-percentage of correct circuits.
- Failures such as crystal defects, masks defects, due to contacts and dust particles –less likely to affect a chip when its area is smaller
- ➢ **Speed**: increasing the operation speed will normally require a larger area
- ▪ Design process- trade-off between speed and area
- ▪ Operation speed is a part of the specification and area should be minimized without violating
- It is a design constraint
- ➢ **Power dissipation**: dissipation of more power leads to too hot and stop working or will need extra cooling
- For special category of applications such as portable equipment powered by batteries, low power consumption is required
- trade off- low power may lead to an increase in the chip area
- ➢ **Design Time:** it is never a goal on its own, but it is an economical activity
- Chip should be made available as soon as possible
- Design costs are common factors , when only a small number of chips needs to be manufactures
- CAD tools- helps to shorten the design time
- ➢ **Testability:** all chips have to be tested before being used in a product, since they may be defective
- Easily testable as testing equipment is expensive
- Time spent to test a single chip- minimal

2. Enumerate and explain the VLSI design methods and technologies.

- Full custom design- maximal freedom
- The designer has the ability to determine the shape of every mask layer for the production of the chip.
- shorter design time
- ▪ Semicustom-Design methods with limited freedom
- Implies the use of gate arrays, standard cells, parameterizable modules or a combination of the three.



- Many manufacturers of integrated circuits make available chips that have all their transistors preplaced in regular patterns
- The designer only needs to specify the wiring patterns (in one or more layers of metal) to interconnect these transistors
  Circuits of this type are called gate arrays

3. Discuss the domains and their hierarchies in VLSI design with Y Chart.

- Three design domains- each with own hierarchy

➢ **The Behavioral Domain**

- A part of the design (or the whole) is seen as a black box
- The relations between outputs and inputs are given without a reference to the implementation of these relations
- A behavioral description at the transistor level- e.g. an equation giving the channel current as a function of the voltages at source, drain and gate or the description of a transistor as ideal switch
- At a higher level- a design unit with the complexity of several transistors can easily be described by means of expressions in Boolean algebra or by means of truth tables
- Register-transfer level- circuit is viewed as sequential logic consisting of memory elements(registers) and functions that compute the next state given the current memory state
- Highest descriptions- Algorithms that may not refer to the hardware that will realize the computation described

➢ **The Structural Domain**

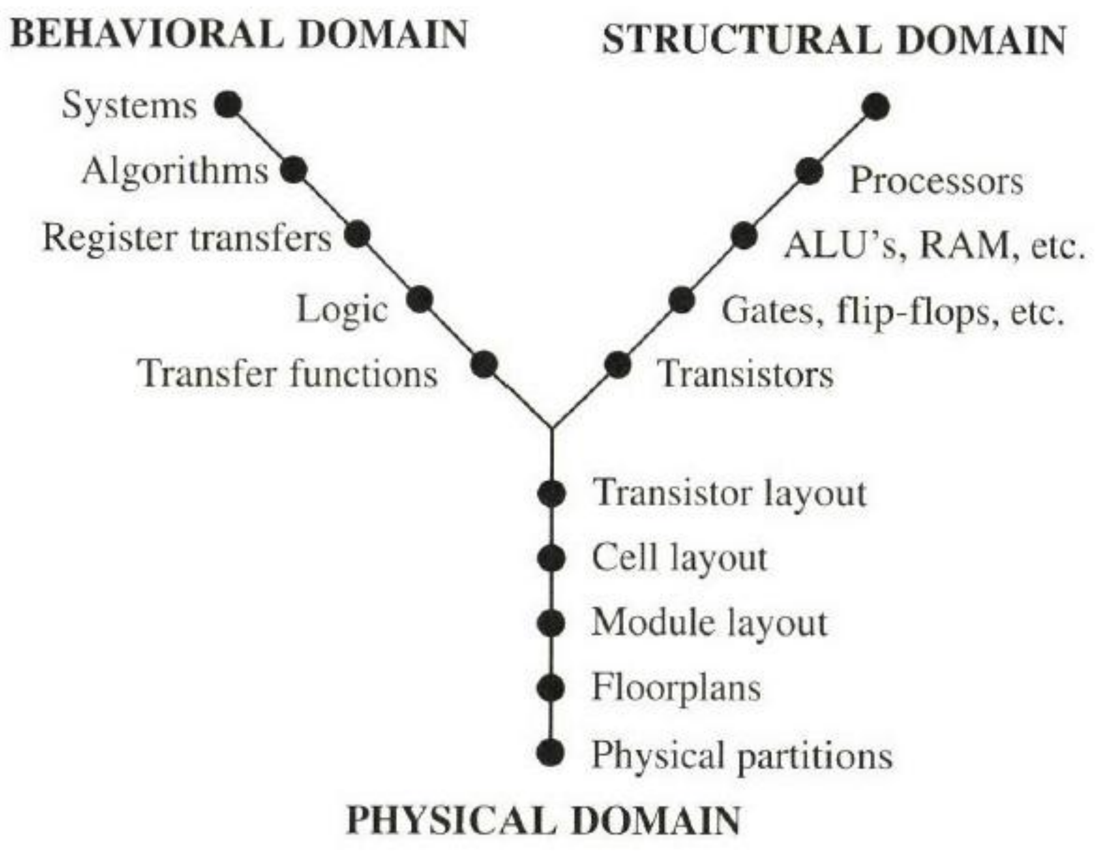
▪ Circuit- composition of subcircuits
▪ Description gives information on the subcircuits used and the method of interconnection

- Each of the subcircuits- description in the behavioral domain or a description in the structural domain itself (or both).

- Example- schematic showing how transistors should be interconnected to form a NAND gate or how this NAND gate can be combined with other logic gates to form some arithmetic circuit

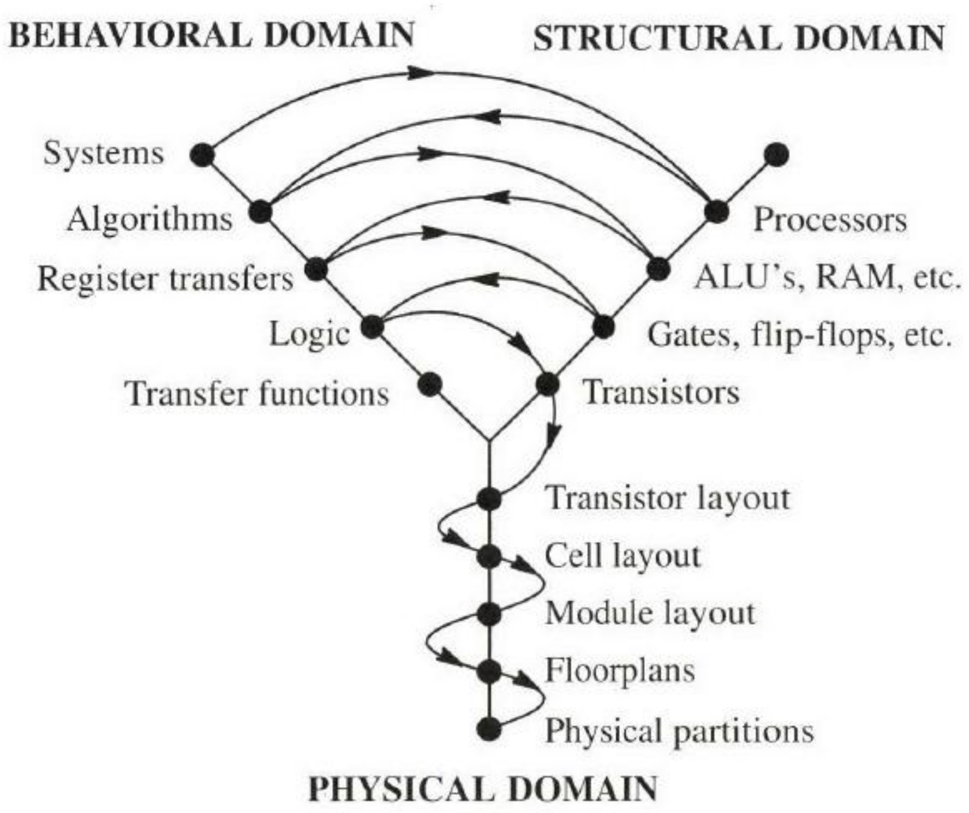
➢ **The Physical (or Layout)domain**

- VLSI Circuit- realized on a chip which is always two-dimensional
- Gives information on how the subparts that can be seen in the structural domain, are located on the two-dimensional plane
- Example- a cell that may represent the layout of a logic gate will consist of mask patterns that form the transistors of this gate and the interconnections within the gate.
- Three domains and their hierarchies- visualized as Y-Chart
- Each axis- represents a design domain and the level of abstraction decreases from the outside to the center
- Gajski Khun Y-Chart
- Y-Chart is a powerful tool to illustrate different design methodologies



- The path drawn in the Y-chart illustrates a top-down design methodology
- Parts with known behavior are decomposed into smaller blocks with simpler behavior and an interconnection structure.
- Corresponds with a transition from the behavioral to the structural domain.
- Each subpart can again be thought to be
- Located on the behavioral axis and is decomposed in its own turn.

- Process continues until a sufficiently low level of abstraction is reached.
- Once the circuit has been fully specified down to the lowest structural level, the layout is specified in a bottom-up fashion
- Transistors are grouped to form cells; cells are grouped to form modules, etc.



4. Explain briefly about the data structures for the representation of Graphs.

- To implement Graph algorithms- Data Structures
- No optimal data structure that should be used in all algorithms. Different algorithms have different requirements.

(i) **Adjacency Matrix**

- Consider a Graph G(V,E) with n vertices
- Adjacency matrix is represented by A
  A_{ij} = 1, if (v_i,v_j) ε E, and A_{ij} = 0 ,if (v_i,v_j) ε E
  As the edges do not have directions in undirected graphs, the adjacency matrix of such graphs is symmetric.

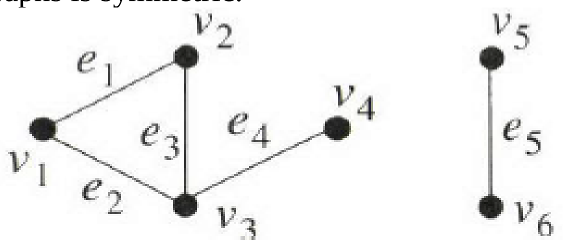
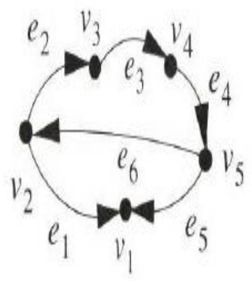


Figure - Undirected Graph

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

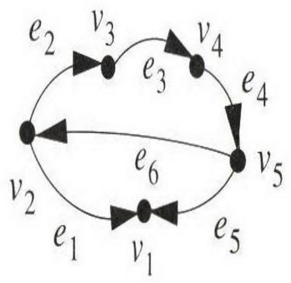
Figure - Adjacency Matrix of Undirected Graph



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure- Directed Graph **Figure - Adjacency Matrix of directed Graph**

- When using the adjacency matrix, testing whether two given vertices are connected or not can be performed in constant time
- The time to perform the test does not depend on the size of the graph.
- But finding all vertices connected to a given vertex require inspection of a complete row and a complete column of the matrix.
- Not very efficient when most of the entries in the matrix are equal to zero.
- The adjacency list, representation is a better choice in such a case.
- Adjacency List- Array that has as many elements as the number of vertices in the graph
- An array element identified by an index i corresponds with the vertex v_i
- Each array element points to a linked list that contains the indices of all vertices to which the vertex corresponding to the element is connected



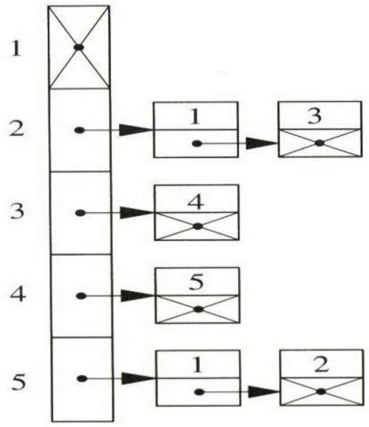


Figure – Directed Graph **Figure -Adjacency List**

- Edges as well as vertices are identified by an integer number
- This number can also be the index in an array pointing to all vertices or all edges
- Each vertex points to the list of edges incident from it by means of the member (field in Pascal) *outgoing-edges* in the *vertex* structure and the member *next* in the *edge* structure.
- Each edge points to the two vertices that it connects by means of the members to and from

```
struct vertex {
  int vertex_index;
  struct edge *outgoing_edges;
};

struct edge {
  int edge_index;
  struct vertex *from, *to;
  struct edge *next;
};
```

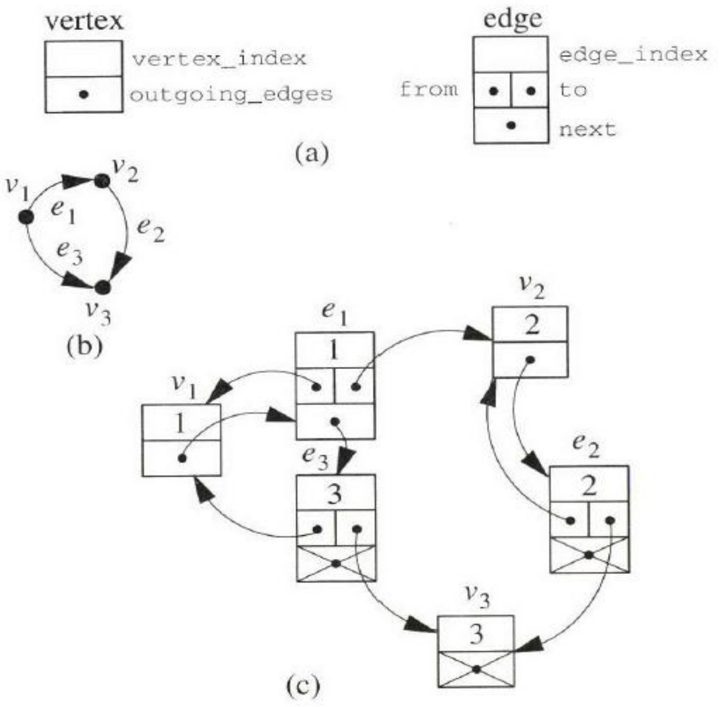


Figure 3.7 The visualization of the vertex and edge structures (a), a small directed graph (b) and its representation with a data structure built from those structures (c).

5. **Explain with a pseudo code and example about the depth first search algorithm**.

   In many graph algorithms- needs to traverse the graph in one way or the other and "do something" with the nodes and/or edges that encounters during the traversal.

   Function that performs the depth-first search is called *dfs* and it is mentioned in the pseudo code.

   The goal is to visit all vertices only once

- ***mark*** in the vertex structure- this member is initialized with the value 1 and given the value 0 when the vertex is visited
- As the value is never restored to 1 and only the vertices whose mark members have value 1 are visited, it is guaranteed that each vertex is visited at most once.
- ***dfs*** - a recursive function that takes a vertex as its argument
- It "processes" the vertex (the generic vertex action) and then inspects all its outgoing edges one by one

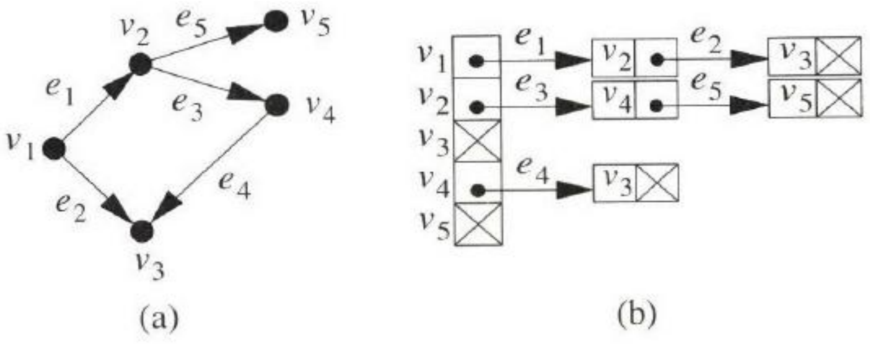
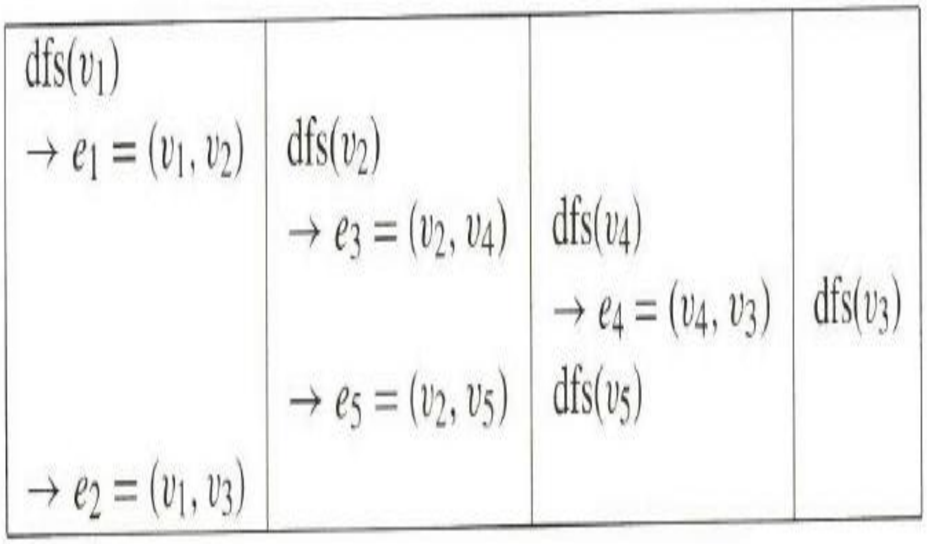


Figure 3.9 A directed graph (a) and its adjacency-list representation (b).



$\text{dfs}(v_1)$			
$\rightarrow e_1 = (v_1, v_2)$	$\text{dfs}(v_2)$		
	$\rightarrow e_3 = (v_2, v_4)$	$\text{dfs}(v_4)$	
		$\rightarrow e_4 = (v_4, v_3)$	$\text{dfs}(v_3)$
	$\rightarrow e_5 = (v_2, v_5)$	$\text{dfs}(v_5)$	
$\rightarrow e_2 = (v_1, v_3)$			

Figure 3.10 The different steps of the depth-first search algorithm applied to the graph of Figure 3.9(a).

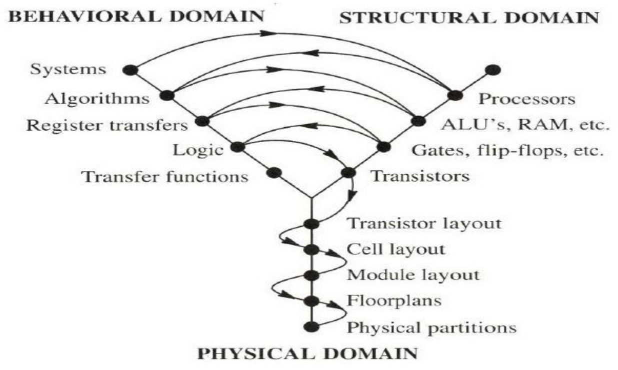
- After "processing the edge" (the generic edge action), the vertex to which this edge is incident is used for a recursive call of ***dfs***, unless the vertex had already been visited.
- In the main program, the function is applied to each vertex of the graph to account for the fact that not all vertices may be reachable from a single vertex.

- Adjacency List representation - appropriate in this situation as it gives direct access to the outgoing edges of each vertex.
- Each vertex is visited exactly once. Since the outgoing edges of a vertex are only visited when the vertex itself is visited, all edges are also visited exactly once
- Time complexity of O(n + lEl) for depth-first search, where n = lVl.
- In the figure, the recursion depth increases from left to right.
- The edges processed as a result of a call to dfs at a specific recursion depth are shown in the same column as the function call (preceded by an arrow)
- Time increases from left to right and from top to bottom
- the vertices are visited in the order v1, v2, v4, v3 and v5 while the edges are visited in the order e1, e3, e4, e5 and, e2.
- Because all vertices are reachable from v1, the first vertex for which ***dfs*** is called, there is only a single call to ***dfs*** at the lowest level of recursion.

6. Draw the Y-Chart showing the VLSI design tools and explain them in detail.

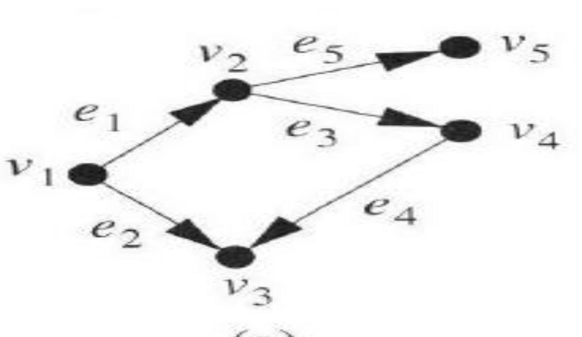
Design actions can clearly be visualized as a transition in the Y-chart either within a single domain or from one domain to another.
Synthesis steps- they add detail to the current state of the design
Synthesis steps may be performed fully automatically by some synthesis tool or manually by the designer
Ideally, a synthesis step is correct by construction, which means that the transformation required to obtain the new description from the old one has the property that it preserves the behavior of the sub circuit involved.

- When synthesis steps do not lead to a result that is correct by construction, there is a necessity to use verification tools.
- Analysis tools- They provide data on the quality of the design (e.g. how fast is the circuit? how large is its area?)
- Hints on how to optimize the design (e.g. which part determines the maximal speed and should receive more attention by the designer?).
- optimization tools- improve the quality of a design without necessarily making a transition to another level of abstraction or design domain
- a logic optimization tool can replace a set of Boolean expressions by an equivalent set which is cheaper to realize
- Design management tools- design data storage, tool communication, the invocation of tools in the right

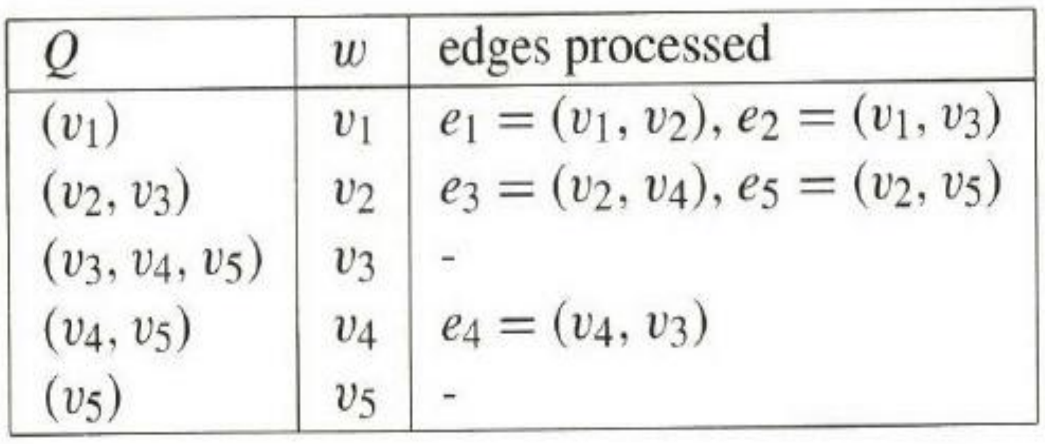


7. Explain with an example the breadth-first search algorithm with a Pseudo-code

- Breadth-first search is an alternative to depth-first search for systematically visiting all vertices of a graph.
- The function that performs the actual breadth-first search is called ***bfs***.
- The central element in the description is the ***FIFO*** queue (first-in first-out queue)***Q***
- Objects can be added and removed from such a queue in such a way that the order in which objects are removed is identical to the order in which the objects were originally added
- The call ***shift_in (q, o)*** adds an object ***o*** to the queue ***q***
- The call ***shift-out (q)*** removes the oldest object from the queue ***q***
- Empty queue is denoted by ( )
- Adding and removing objects from a FIFO queue can be done in constant time.
- The main difference with depth-first search is that calling the ***dfs*** function recursively with a new vertex adjacent to the current vertex has been replaced by the addition of the new vertex to the ***FIFO queue***.
- The two methods visit the vertices and edges in a different order, although all vertices and edges are eventually visited by both methods
- Algorithm visits all vertices and all edges of a graph exactly once leading to a time Complexity of ***O(n + lEl)***
- Changing the behavior of the queue from FIFO to LIFO (last in first out) changes the breadth-first algorithm into the depth-first algorithm
- The application of ***bfs*** to the graph of Figure (a) shown below is illustrated in Figure (b)



(a)



Q	w	edges processed
(v_1)	v_1	$e_1 = (v_1, v_2), e_2 = (v_1, v_3)$
(v_2, v_3)	v_2	$e_3 = (v_2, v_4), e_5 = (v_2, v_5)$
(v_3, v_4, v_5)	v_3	-
(v_4, v_5)	v_4	$e_4 = (v_4, v_3)$
(v_5)	v_5	-

Figure (b)

- Each line of the figure corresponds to one iteration of the do loop in the pseudo-code description
- For each iteration, the contents of the FIFO queue (the leftmost object is the oldest), the vertex processed and the edges processed are given
- The edges processed determine the vertices added to the queue.
- From the figure it can be concluded that the vertices are visited in the order v1, v2, v3, v4, v5 while the edges are visited in the order e1, e2, e3, e5 and e4.
- Breadth-first search has an additional property.
- First all vertices adjacent to Vs are visited.
- These are the vertices connected to Vs, with a path of length 1.
- The vertices visited next are reachable from Vs, through a path of length 2 , etc
- vertices are visited in the order of their shortest path from Vs.

```
/* Given is the graph $G(V, E)$ */

struct vertex {
  . . .
  int mark;
};

bfs(struct vertex $v$)
{
  struct fifo *$Q$;
  struct vertex $u$, $w$;
  $Q \leftarrow ()$;
  shift_in($Q$, $v$);
  do { $w \leftarrow$ shift_out($Q$);
         "process $w$";
         for each $(w, u) \in E$ {
           "process $(w, u)$";
           if ($u$.mark) {
             $u$.mark $\leftarrow 0$;
             shift_in($Q$, $u$);
           }
         }
  } while ($Q \neq ()$)
}

main ()
{
    for each $v \in V$
      $v$.mark $\leftarrow 1$;
    for each $v \in V$
      if ($v$.mark) {
        $v$.mark $\leftarrow 0$;
        bfs($v$);
      }
}
```